Course Name: Advanced Java

Lecture 4
Topics to be covered

- Arrays
- Classes & Methods,
- Inheritance

INTRODUCTION TO ARRAYS

The following variable declarations each allocate enough storage to hold one value of the specified data type.

```
int number;
double income;
char letter;
```

An **array** is an object containing a list of elements of the **same** data type

Arrays

We can create an array by:

- Declaring an array reference variable to store the address of an array object.
- Creating an array object using the `new` operator and assigning the address of the array to the array reference variable.

Here is a statement that declares an array reference variable named *dailySales*:

```
double[ ] dailySales;
```

The brackets after the key word `double` indicate that the variable is an array reference variable. This variable can hold the address of an array of values of type `double`. We say the data type of *dailySales* is `double` array reference.

The second statement of the segment below creates an array object that can store seven values of type `double` and assigns the address of the array object to the reference variable named "*dailySales*":

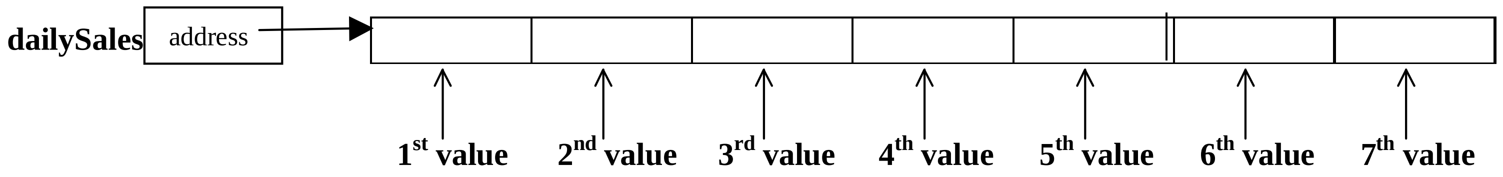
```
double[ ] dailySales;

dailySales = new double[7];
```

- The operand of the `new` operator is the data type of the individual array elements and a bracketed value that is the array size declarator. The **array size declarator** specifies the number of elements in the array.

- It is possible to declare an array reference variable and create the array object it references in a single statement.
- The statement below creates a reference variable named *dailySales* and an array object that can store seven values of type `double` as illustrated below:

Here is an example:

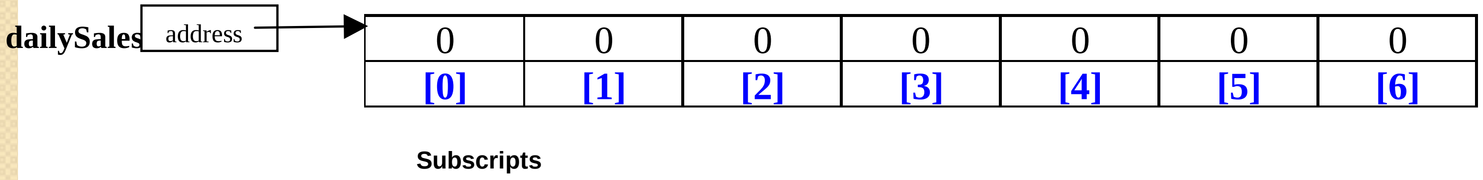
```
double[ ] dailySales = new double[7];
```



Accessing Array Elements

- We can access the array elements and use them like individual variables.
- Each array element has a subscript. This subscript can be used to select/pinpoint a particular element in the array.
- **Array subscripts are offsets from the first array element**.
- The first array element is at offset/subscript 0, the second array element is at offset/subscript 1, and so on.
- The subscript of the last element in the array is one less than the number of elements in the array.

```
final int DAYS = 7;

double[ ] dailySales = new double[DAYS];
```



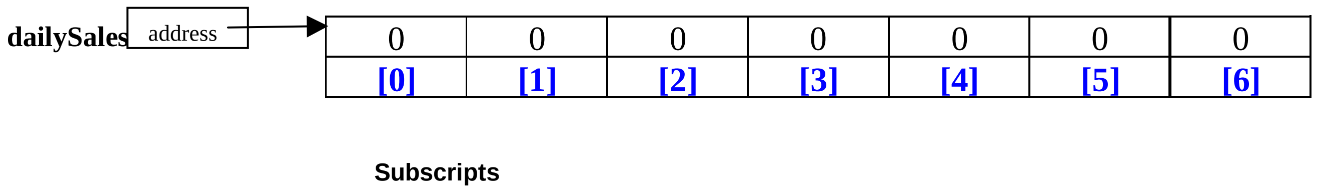
dailySales[0], pronounced *dailySales* sub zero, is the first element of the array.

dailySales[1], pronounced *dailySales* sub one, is the second element of the array.

dailySales[6], pronounced *dailySales* sub six, is the last element of the array.

- Array subscripts begin with **zero** and go up to *n - 1*, where *n* is the number of elements in the array.

```
final int DAYS = 7;

double[ ] dailySales = new double[DAYS];
```



Typically, we use a loop to cycle through all the subscripts in the array to process the data in the array.

Array Initialization

- Like other variables, you may give array elements an initial value when creating the array.

  Example:

  The statement below declares a reference variable named *temperatures*, creates an array object with room for exactly tens values of type `double`, and initializes the array to contain the values specified in the initialization list.

```
double[ ] temperatures = {98.6, 112.3, 99.5, 96, 96.7, 32, 39, 18.1, 111.5};
```

- By default, Java initializes the array elements of a numeric array with the value 0.

```
int[ ] attendance = new int[5] ;
```

Array Length

- Each array object has an attribute/field named *length*. This attribute contains the number of elements in the array.

  For example, in the segment below the variable named *size* is assigned the value 5, since the array referenced by *values* has 5 elements.

```
int size;
int[ ] values = {13, 21, 201, 3, 43}

size = values.length;
```

Notice, *length* is an attribute of an array not a method - hence no parentheses.

class definition

```
class classname {
    field declarations
    { initialization code }
    Constructors
    Methods
}
```

Inheritance

- On the surface, inheritance is a code re-use issue.
  - we can *extend* code that is already written in a manageable manner.
- Inheritance is more
  - it supports polymorphism at the language level

Inheritance

- The derivation of one class from another class is called Inheritance.
- Types of inheritance

Inheritance

- One object type is defined as being a special version of some other object type.
  - a *specialization*.
- The more general class is called:
  - base class, super class, parent class.
- The more specific class is called:
  - derived class, subclass, child class.

- A class that is inherited is called a superclass.
- The class that does the inheriting is called as subclass.
- In above figure all class A is superclass.
- A subclass inherits all instance variables and methods from its superclass and also has its own variables and methods.
- One can inherit the class using keyword extends.

- Syntax :

```
Class subclass-name extends superclass-name
{
        // body of class.
}
```

- In java, a class has only one super class.
- Java does not support Multiple Inheritance.
- One can create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass. However, no class can be a superclass of itself.

Example

```
class A                                     //superclass
{    int num1;                              //member of superclass
     int num2;                              //member of superclass
     void setVal(int no1, int no2)          //method of superclass
     {         num1 = no1;
               num2 = no2;
     }
}
class B extends A //subclass B
{    int multi; //member of subclass
     void mul() //method of subclass
     {         multi = num1*num2; //accessing member of superclass from subclass
     }
}
class inhe2
{    public static void main(String args[])
     {         B subob = new B();
               subob.setVal(5,6);       //calling superclass method through subclass object
               subob.mul();
               System.out.println("Multiplication is " + subob.multi);
     }
}
```

Output : Multiplication is 30

Note

- Private members of superclass are not accessible in sub class
- Superclass is also called parent class or base class,
- subclass is also called child class or derived class.